Relatório de Refatoração e Análise de Bad Smells

Disciplina: Teste de software
Professor: Cleiton Tavares
Trabalho: Refatoração e Análise de Bad Smells em JavaScript
Nome: Henrique Lobo Leite Neves
Matrícula: 808840

1. Análise de Smells

1.1. Long Method (Método Longo)

O método `generateReport` no código original é excessivamente longo (mais de 60 linhas) e realiza múltiplas responsabilidades. Este smell é problemático porque:

- Dificulta a compreensão do código, pois mistura várias preocupações diferentes
- Torna os testes mais complexos, pois há muitos casos de teste para um único método
- Aumenta a chance de bugs, pois alterações em uma parte podem afetar outras inadvertidamente
- Viola o princípio de responsabilidade única (SRP)

Exemplo do código original:

```
generateReport(reportType, user, items) {
    let report = '';
    let total = 0;
    // ... mais de 60 linhas de código com múltiplas responsabilidades
}
```

1.2. Cognitive Complexity (Complexidade Cognitiva)

O código original possui uma estrutura de controle altamente aninhada com múltiplos `if/else` encadeados. O ESLint/SonarJS detectou uma complexidade cognitiva de 27 (muito acima do limite recomendado de 15). Isto é problemático porque:

- Torna o código difícil de entender e manter

- Aumenta a probabilidade de erros lógicos
- Dificulta a adição de novos tipos de relatórios ou regras de usuário
- Torna os testes mais complexos devido aos múltiplos caminhos de execução

1.3. Code Duplication (Duplicação de Código)

O código original repete várias estruturas similares, especialmente no cálculo de totais e na formatação de relatórios. Problemas causados:

- Violação do princípio DRY (Don't Repeat Yourself)
- Aumento do risco de bugs quando uma das cópias é atualizada mas outras não
- Maior esforço de manutenção quando mudanças são necessárias
- Dificuldade em garantir consistência entre as diferentes implementações

2. Relatório da Ferramenta

Resultado da análise do ESLint/SonarJS no código original:

```
C:\repositorios\bad-smells-js-refactoring\src\ReportGenerator.js
  11:3   error  Refactor this function to reduce its Cognitive Complexity
from 27 to the 5 allowed  sonarjs/cognitive-complexity
  43:14  error  Merge this if statement with the nested one
sonarjs/no-collapsible-if

✖ 2 problems (2 errors, 0 warnings)
```

O eslint-plugin-sonarjs foi particularmente útil em:

1. Identificar precisamente a complexidade cognitiva (27) do método principal
2. Detectar estruturas condicionais que poderiam ser simplificadas
3. Fornecer métricas objetivas para qualidade do código
4. Identificar oportunidades específicas de melhoria

3. Processo de Refatoração

O smell mais crítico tratado foi a alta complexidade cognitiva. A principal técnica utilizada foi a combinação de:

1. Extract Method
2. Replace Conditional with Polymorphism
3. Template Method Pattern

Antes:

```
generateReport(reportType, user, items) {
    let report = '';
    let total = 0;

    if (reportType === 'CSV') {
        report += 'ID,NOME,VALOR,USUARIO\n';
    } else if (reportType === 'HTML') {
        report += '<html><body>\n';
        report += '<h1>Relatório</h1>\n';
        // ... mais condicionais aninhados
    }

    for (const item of items) {
        if (user.role === 'ADMIN') {
            if (item.value > 1000) {
                item.priority = true;
            }
            // ... mais condicionais aninhados
        }
    }
    // ... mais código
}
```

Depois:

```
class ReportGenerator {
    constructor(database) {
        this.db = database;
        this.formatters = {
            CSV: new CSVFormatter(),
            HTML: new HTMLFormatter()
        };
    }

    generateReport(reportType, user, items) {
        const formatter = this.formatters[reportType];
        const filteredItems = this.filterItemsByUserRole(items, user);
        const processedItems = this.processItems(filteredItems, user);
        const total = this.calculateTotal(processedItems);
```

```
        return formatter.generateReport(processedItems, user, total);
    }

    filterItemsByUserRole(items, user) {
        if (user.role === 'ADMIN') return items;
        return items.filter(item => item.value <= this.PERMISSION_THRESHC
    }
}
```

4. Conclusão

A utilização de testes como rede de segurança foi fundamental durante o processo de refatoração. Os testes nos permitiram:

1. Garantir que a funcionalidade original foi preservada
2. Refatorar com confiança, sabendo que quebras seriam detectadas
3. Validar cada alteração incrementalmente
4. Manter a qualidade do código durante todo o processo

A redução dos bad smells resultou em:

- Código mais modular e fácil de manter
- Melhor separação de responsabilidades
- Maior facilidade para adicionar novos formatos de relatório
- Redução significativa da complexidade cognitiva
- Maior reusabilidade de código
- Melhor testabilidade

O uso combinado de análise manual, ferramentas automatizadas (ESLint/SonarJS) e uma sólida suite de testes proporcionou um processo de refatoração seguro e efetivo, resultando em um código mais limpo e sustentável.

Esta experiência demonstra a importância de identificar e corrigir bad smells de forma sistemática, sempre apoiado por testes automatizados e ferramentas de análise estática de código.